Code Review by Darian and Kai

Functions to be Refactored:
1. **public void spawning() in all Characters Classes**
Smells: Duplicated code: The code block inside the while loop is duplicated.
Long method: The spawning() method does more than one thing. It checks for collision and also handles relocation.
Inefficient code: The random number generation code can be optimized.
Solutions: Duplicated code: The duplicated code has been moved to a new method called "relocate()", which is now called from the while loop and the spawning() method.
Long method: The spawning() method has been refactored to only handle spawning, and a new method called "relocate()" has been created to handle relocation. This separates the concerns of the two actions and makes the code more readable and maintainable.
Conclusion: Extracted the duplicated code into a helper method called relocate() that generates the randomly spawned coordinates for the characters objects.

2. **public void run() in Game Engine Class**
Smells: Long method: The original run method was quite long and had multiple responsibilities, such as handling the game loop, updating objects, and sleeping the thread.
Complicated and Long numbers: The original code contained magic numbers, such as 60, 1000000000, 0.4, and 0.2, which can make the code hard to read and maintain.
Conditional complexity: The original code had nested conditional statements and multiple boolean checks within a single condition, making it hard to read and understand.
Solutions: Long method: Extracting methods can help to break down the original long method into smaller and more manageable pieces, with each method having a single responsibility. Constants: Introducing constants can help to give names to magic numbers, making the code more readable and easier to maintain. Such as, using _ in big numbers.
Variable Names: Using consistent variable names throughout the code can help to make the code more readable and debuggable.
Conditionals: Simplifying the conditionals can make the code more readable and easier to understand.
Conclusion: The refactored code extracts three methods that handle updating the enemies, updating time-sensitive objects, and sleeping the thread, respectively. This helps to reduce the size of the original method and improve its readability. The conditional complexity is simplified by introducing a separate method to check if any movement keys are pressed. The refactored code is more readable, maintainable, and easier to understand.

3. **public void draw(Graphics2D g) in UI Class**
Smells: Code Duplication: There is a significant amount of duplication in the code, with similar code blocks repeated for the victory and game over screens.
Long Method: The draw method is quite long and does multiple things: it draws the score and time info, as well as the end game screens.

Solutions: Code Duplication: Extract the duplicated code into a helper method that can be called from both places.
Long Method: Split the method into smaller, more focused methods that each do one thing. In this case, we created a method that draws the end game screens.
Conclusion: Extracted the duplicated code into a helper method called drawEndScreen that takes the necessary parameters for drawing the end screen, and calls it from both places. We used more descriptive variable names. Reorganized the code to improve readability and maintainability.

4. **public void getCellImage() in GameWorld Class**
I have refactored my public void getCellImage() in the GameWorld because all the resources and the order number were messy. Therefore I reordered the order number from the most to the least used images. There were many blanks between some of the numbers and unused image numbers, so it went from 0 to 40, but after I refactored this function, it became 0 to 30, which looks simpler. Also, I got more space to add images when I want to update. I have also recreated the map.txt files, and since I used most used images as 1-digit numbers, it looks more organized, and I can save more time when I want to edit the text.File.

5. **public void update(GameInput keyBoard) in Main Character Class**
I have extracted some code into separate methods for clarity and reusability. For example, I made updatePosition(distance) and updateSpriteMovement() methods and put the original codes in there. I used the Used a ternary operator for the spriteMovement update. I reordered the methods for better readability.

6. **public void drawMap(String filePath, int map) in GameWorld**
I replaced the while loop with a for loop, which makes the code more concise and easier to read—also, adding a for loop for rows to Handle the case where the file has fewer rows than gameBarrier.maxScreenRow. Overall, these changes make the code more modular, easier to read, and less error-prone.

7. **public void draw(Graphics2D g2d, int cellSize) in GameWorld**
I reduced the number of variables: In the original code, there were four variables (col, row, x, y) that were being used to keep track of the current position while drawing the map. By using a nested for-each loop, I could eliminate the need for col and row, and by using a single loop and calculating the position based on the loop variable, I could eliminate the need for x and y. Reduced the chance of errors: The original code had the potential for errors due to using multiple variables to keep track of the current position, which could get out of sync if not properly updated. By using a for-each loop and calculating the position based on the loop variable, there is less chance of this happening. Overall, these changes make the code more readable, more concise, and less error-prone, which can help make it easier to maintain and modify in the future.